the I.P.Sharp newsletter

March/April 1980
Vol. 8 Number 2

# Important System Changes

## CHANGES TO THE BEHAVIOUR OF BRANCH AND ⎕LC

Leslie Goldsmith, Toronto, and Paul Berry, Palo Alto

**Effective 3 May 1980, the SHARP APL system will revise its treatment of certain classes of branch, and will also revise the way it represents execute (⍎) and quad (⎕) in the line counter, ⎕LC.**

These changes will help to achieve the following goals:

1. The definition of branch will be simple and consistent. The behaviour of branch will no longer depend on whether it was invoked from a function, from immediate execution, from ⎕TRAP, or from ⍎. The meaning of →0 will no longer depend upon its context; similarly, the meaning of →⍳0 will no longer depend upon its context. After the changes, the statements →*X*, ⍎'→*X*', and →⍎'*X*' will behave identically (whereas at present they don't always).

2. Branch from within a statement executed via ⍎ (which is how the latent expression ⎕LX is executed) will behave in a way that is compatible with **both** the current SHARP APL system **and** IBM APL systems.

## IN THIS ISSUE

Changes to Branch and ⎕LC ..... 1
On-line APL Bibliography ..... 5
Sharp User Meeting in October ..... 6
Air Transporation Seminar, London ..... 7
New Options Data ..... 8
Course Schedule ..... 9

**Technical Supplement 25**
Branching Part 4 ..... T1
Contest 9 Results ..... T6

## Changes Affecting Transfer of Control

The list below summarizes the changes affecting branch operations. The behaviour of → (so called 'naked branch') is not affected by these changes:

1. Any branch (other than → alone) typed from within ⎕ mode will result in a *SYNTAX ERROR*. (At present, only →0 and →⍳0 are permitted, and ⎕ returns the values 0 or ⍳0 respectively.)

2. All branches of the form *R*←⍎'→*X*' will result in a *SYNTAX ERROR*. (At present, this is permitted when *X* is 0 or ⍳0, and *R* is assigned the value 0 or ⍳0 respectively, but no other value of *X* is permitted.)

3. An immediate-execution line or ⎕*TRAP* recovery expression ending with →⍳0 will behave as if the branch were not there. (At present, such a branch causes the function highest on the state indicator to resume at its next line, ⎕*LC*+1.)

4. →0 from within execute will terminate the function highest on the state indicator. (At present, ⍎'→0' terminates execution of the argument of ⍎, but has no effect on the encompassing function.)

## Changes Affecting the Line Counter

The contents of the line counter, ⎕*LC*, will change slightly for elements that represent ⍎ or ⎕. In SHARP APL (as in VS APL, but not as in APLSV), each use of ⍎ and ⎕ has been represented by its own row in the state indicator. Further, in SHARP APL each use of ⍎ or ⎕ has its own element in ⎕*LC*; each such element has had the value zero. (By contrast, in both VS APL and APLSV, ⎕*LC* contains **no** elements corresponding to ⍎ or ⎕; hence, in VS APL, ⎕*LC* may have fewer elements than there are rows in the state indicator.)

After the change, the state indicator will remain as it is, and ⎕*LC* will continue to have an element corresponding to each level of the state indicator. However, the value of each element of ⎕*LC* that corresponds to a use of ⍎ or ⎕ will be the **line number of the defined function that invoked it**, rather than zero. When ⍎ or ⎕ is used in immediate execution mode with a suspended function on the state indicator, ⎕*LC* will contain the line number on which that function was suspended. Only when ⍎ or ⎕ is used in immediate execution mode with **no** defined function on the state indicator will ⎕*LC* contain a 0.

## Impact of the Changes

The primary effect of the change to ⎕*LC* is as follows. To provide automatic restarting of an interrupted task, the latent expression ⎕*LX* must contain a branch. However, when the system executes ⎕*LX*, it does so (now and after the change) by invoking ⍎, and therefore ⍎ goes on the state indicator, and ⎕*LC*'s first element corresponds to that ⍎. Because the first element of ⎕*LC* describes the system's use of ⍎ (rather than the function suspended in the workspace), in SHARP APL as it is at present, a latent expression to resume execution where work was interrupted must use →1↓⎕*LC* rather than →⎕*LC*. The 1↓ is there to dispose of the leading zero in the line counter. By contrast, programs running in IBM APL systems must use →⎕*LC* instead.

After the change, →⎕*LC* and →1↓⎕*LC*, when they occur in the latent expression or in any other argument to ⍎, will have the same effect. Latent expressions prepared for the present Sharp system and those prepared for IBM systems will both run correctly. The change will **not** affect the operation of existing programs running on SHARP APL, unless a program explicitly assumes that, within ⍎ or ⎕, 1↑⎕*LC* is 0.

The new behaviour of branch is summarized in the table below. Entries that differ from the current behaviour are highlighted with shading. *NZ* represents any integer other than 0.

**New Behaviour of Branch**

| | →*NZ* | →⎕*LC* | →0 | →⍳0 |
|---|---|---|---|---|
| 1. Fn | Current fn goes to line *NZ* | Current fn goes to line now on | Current fn exits | No effect |
| 2. Imm-ex or trap line | Fn highest on SI resumes at line *NZ* | Fn highest on SI resumes on suspended line | Fn highest on SI exits | No effect |
| 3. ⍎'→*X*' | Fn highest on SI goes to line *NZ* | Fn highest on SI goes to line now on | Fn highest on SI exits | No effect |
| 4. *R*←⍎'→*X*' | *SYNTAX ERROR* | *SYNTAX ERROR* | *SYNTAX ERROR* | *SYNTAX ERROR* |
| 5. ⎕ | *SYNTAX ERROR* | *SYNTAX ERROR* | *SYNTAX ERROR* | *SYNTAX ERROR* |

Most uses of branch (shown in the first column of the table) will not be affected at all. Branching to a nonzero integer (such as a line label) in a function, ⍎, ⎕*TRAP*, or immediate execution, will continue to behave exactly as it does at present.

Of the four categories of branches whose operation will be affected, the first two categories (branches within ⎕ mode, and branches within execute where a result is expected) are esoteric cases which are exceedingly unlikely to occur.

Branches belonging to the third category (→⍳0 in immediate execution or in ⎕*TRAP*) are also unlikely to occur. ⎕*TRAP* expressions which make use of the present behaviour of →⍳0 should be changed to perform →⎕*LC*+1 instead.

The final category of branch changes (→0 within execute) is the one most likely to affect the behaviour of existing programs. Several simple techniques that are unaltered by the upcoming changes may be used to convert existing statements. The statements suggested here will work both before the change and after it.

As an example, suppose the following line of code is used in a program:

```
[4]   CN←CN+1 ◇ ⍎PRFLG/'J←0 ◇ →(×100|CN)ρ0 ◇ CN' ◇ K←1↓K
```

The line first adds 1 to *CN* each time through; then, if *PRFLG* is 1, it executes an expression which sets *J* to 0, and prints the value of *CN* if it is a multiple of 100. The last statement on the line, which is always executed, discards the first element of *K*.

An equivalent line which uses nested executes to achieve the same effect that the computed →0 above does, can be constructed as follows:

```
[4]   CN←CN+1 ◇ ⍎PRFLG/'J←0 ◇ ⍎(0=100|CN)/''CN''' ◇ K←1↓K
```

An alternate method makes use of separate lines and an additional line label, with the same effect:

```
[4]   CN←CN+1 ◇ ⍎PRFLG/'J←0 ◇ →(×100|CN)⍴L0 ◇ CN'
[5]  L0:K←1↓K
```

Still another method involves removing the use of execute altogether—often the best choice, since execute is frequently overused in cases where straightforward conditional branch logic would be both easier to read and more efficient:

```
[4]   CN←CN+1 ◇ →PRFLG↓L0 ◇ J←0 ◇ →(×100|CN)⍴L0 ◇ CN
[5]  L0:K←1↓K
```

Even simpler coding transformations may exist, depending upon the exact nature of the line. For example, if the original line above did not contain the assignment of *J* to 0, then an easier conversion that involves the same number of executes and no new lines would change it to:

```
[4]   CN←CN+1 ◇ ⍎(PRFLG>100|CN)/'CN' ◇ K←1↓K
```

A simplification of the new behaviour of ⍎'→0' is that it now becomes straightforward to cause the function invoking an execute to terminate from within the execute. For example:

```
[3]   ⍎(2<⍴⍴X)/'⎕UNTIE TN ◇ →0'
```

With the system's current behaviour, the self-explanatory →0 does not have the desired effect, and you must instead replace it by a branch to ¯1 or some other nonzero value for which there is no line in the function.

## A Final Comment

Users who make use of any of the branch facilities that will change, notably →0 in execute, should attempt to locate and alter all occurrences of them **as soon as possible**. Please don't leave this until the last minute, as you may forget and find that, after 3 May 1980, your programs no longer function as you had anticipated. The staff of I.P. Sharp Associates will be pleased to answer any specific questions you may have, and to assist you in the conversion task.

## AT LAST — AN ON-LINE APL BIBLIOGRAPHY!

Rosanne Wild, Toronto

"I went to an APL conference a few years ago, and would really like to get my hands on a copy of the paper I heard presented on linear regression. Do you know the one I mean?"

This example, typical of the kind of questions I've been asked over the years by people in the APL community, indicated the need for an on-line APL bibliography. Using a record handling package called MABRA, I entered references for APL literature into the Sharp system. What evolved is a data base of over 1,000 entries—a bibliography of articles, papers, reports and textbooks about APL and its applications.

The use of MABRA for this data base makes it easy for you to search on any of the attributes—author, title, date, type of publication, or subject. MABRA also allows you to search for words or names contained in any of those fields. If you want to find a paper on linear regression, you would do the following (shaded areas show your input):

```
      )LOAD 11 BIBLIOGRAPHY
SAVED 23.29.12 11/11/79
* TYPE DESCRIBE OR START
      START

* DATA MODE, SYSTEM IS 02: APL BIBLIOGRAPHY

02: LR                                          ⍝ To list a record

* LIST RECORDS

* CONSTRAINTS: TITLE⊃REGRESSION                 ⍝ Guessing that the title
    2 FOUND                                     contains ('⊃') the word 'regression'

* PRINT FIELDS: AUTHOR,TITLE

* SORT FIELDS: AUTHOR

* ALIGN PAPER, TYPE SPACE, CR TO START.

AUTHOR                  TITLE

SMILLIE K.W.            REGRESSION ANALYSIS: THEORY AND COMPUTATION
NEWPECK F.F./ABRANOVIC W.A.  A CONVERSATIONAL REGRESSION PACKAGE FOR
                        EXPLORATORY DATA ANALYSIS

* CONSTRAINTS: (TITLE=REGRESSION ANALYSIS)∧(AUTHOR=SMILLIE)  ⍝ To list more details
    1 FOUND                                                  on Smillie's paper

* PRINT FIELDS: SOURCE,REFERENCE,DATE

* SORT FIELDS: space, carriage return

* ALIGN PAPER, TYPE SPACE, CR TO START.
```

| *SOURCE* | *REFERENCE* | *DATE* |
|---|---|---|
| *APL76: CONFERENCE PROCEEDINGS* | *P.* 401 | 22/09/76 |

On-line assistance is available in 11 *BIBLIOGRAPHY* to guide you in using the system. *HELP* is a valid response to any prompt and will result in a brief description of what user input is required. If you would like more detailed information on MABRA, a manual will soon be available from your local Sharp office.

APL usage has grown tremendously in the last several years, and the number of papers, articles, and textbooks dealing with the language and its applications has also increased. I need everyone's help to make sure that this workspace contains interesting and useful material. (Note: I do use some personal discretion as to the items I enter, based on their length and content.) Please send me information on any APL literature that you can't find in 11 *BIBLIOGRAPHY* and that you feel should be there. Your help and comments will be greatly appreciated.

## 1980 APL USERS MEETING

Plans are now into full swing for our second APL Users Meeting. The conference promises to be interesting and informative with an agenda including over 60 speakers from the APL community. A brochure outlining the session topics and speakers, along with the registration form, is now available. If you haven't received it in the mail, please ask your local Sharp office for one, or call our conference coordinator, Rosanne Wild in Toronto at (416) 364-5361.

Some important details:

| | |
|---|---|
| Dates: | October 6-8, 1980 |
| Location: | Hotel Toronto<br>Toronto, Ontario |
| Registration Fee: | $155 U.S., $180 Canadian, or £80 Sterling<br>(Includes a copy of the conference proceedings and admittance to all day sessions and evening courses.) |

## 1980 WORLDWIDE AVIATION SEMINAR SPONSORED BY SHARP IN LONDON

Another worldwide Aviation Seminar hosted by I.P. Sharp Associates will be held on April 10th, 1980 at the Sheraton Skyline Hotel in London, England. This is the fourth such seminar to be sponsored by our company; two have been held in Newport Beach, California, and one in Washington, D.C. If you are interested in attending this seminar, please contact our London office at (01) 730-0361, or our Toronto Corporate Headquarters at (416) 364-5361. There is no fee for the seminar, however all attendees must be pre-registered. The program starts at 9:30 am, there is a complimentary lunch, and a cocktail party when the sessions wrap up at 7:00 pm.

The seminar content is directed at professionals in the aviation community in North America and Europe—air carriers, manufacturers, investment analysts, educational institutions, and government agencies concerned with air transportation analyses.

An outstanding program has been put together, featuring speakers from Trans World Airlines, Pan American World Airways, Air Canada, Air California, Rolls Royce Aero Engines, Continental Airlines, Air France, and the Association of European Airlines. They will discuss such topics as traffic and route analysis, forecasting international markets, aviation data bases, management reporting, air cargo studies, and other subjects relevant to the application of aviation data.

The agenda:

**Introduction**, Ian Sharp, President, I.P. Sharp Associates

**International Market and Route Analysis**, Roger Davis, Manager, Route Analysis, Pan American World Airways

**Management Reporting Requirements for Air California**, Ted Shown, Director of Market Development, Air California

**Les Liaisons Compromises (Les routes aériennes à l'intérieur des etats unis et de l'europe mises en parallèle)**, Brian Loughran, Senior Economist, Association of European Airlines

**Applications for Aviation Data Bases in the Airline Industry**, David Keith, Manager, Aviation Data Bases, I.P. Sharp Associates

**Appréciation des performance des compagnies aériennes à partir des données AEA: méthode d'approche et interprétation**, Ann Marie Ellie, Chef du Service Environment, Conjoncture et Statistiques, Air France

**U.S. Trunks—Indicators and Projections, January 1980**, Judith Driver, Market Research Officer, Rolls Royce Aero Engines

**Airline Planning Systems—Organisation and Content**, James Longley, Manager, Corporate & Marketing Planning Systems Development, Continental Airlines

**Applications in Schedule Analysis with Reference to the OAG Data Base**, Robert Dabrowski, Production Manager OAG, I.P. Sharp Associates

**North Atlantic Traffic Analysis**, Michael Philburn, Director, Route Development, Trans World Airlines

**International Markets through the 1990's**, John Gradek, International New Products Strategy Manager, Air Canada

## NEW OPTIONS DATA

**175 Key U.S. Financial Market Indicators are now available in the U.S. Options Data Base.**

```
  1 American Stock Exchange price index
  4 Dow-Jones indices
  7 NASDAQ Over the Counter (OTC) indices
  5 New York Stock Exchange indices
121 Standard & Poor's indices
  1 Currency Exchange Rate U.S.$ for $100 Cdn.
 19 Moody's Bond Yield averages
  6 Salomon Bond indices
  1 Toronto TSE 300 Composite index
  4 Value Line (VL) indices
  6 Kuhn-Leob Bond indices
---
175
```

Depending on the index type and frequency of its calculation the following facts are available: High index, Low index, Closing index, Index Date, Earnings per Share (EPS), EPS Date, Indicated annual dividend. Four static facts are also available: Market indicator description, Market indicator symbol (left justified), Market indicator symbol (formatted), Market indicator 'Add Date'.

Access is available with either workspace 55 *RETRIEVE* or 39 *MAGIC*. The following is part of the list of all 175 indicators using 55 *RETRIEVE*.

```
      ΔVIEW 'OPTIONUS' ◇ ΔRESULTTYPE 'GLO'      ⍝ set view and data result type
      (MKTIND←'OBJ' ΔSEARCH 'K///') ΔGET 11     ⍝ search for market indicator symbols,
                                                ⍝   get descriptions
      ('7A1,X2,30A1') ⎕FMT (MKTIND;OPT11)       ⍝ display symbols and descriptions

KAMEX    AMERICAN STK EXCHANGE PRICE IN
KDOWI    DOW-JONES INDUSTRIALS 30 STOCK
KDOWR    DOW-JONES TRANSPORT   20 STOCK
KDOWU    DOW-JONES UTILITIES   15 STOCK
KDOWC    DOW-JONES COMPOSITE   65 STOCK
KNASD    NASDAQ OTC INDEX      COMPOSIT
KBANK    NASDAQ OTC INDEX      BANKS
KFINL    NASDAQ OTC INDEX      FINANCIA
KINDS    NASDAQ OTC INDEX      INDUSTRI
KINSR    NASDAQ OTC INDEX      INSURANC
KTRAN    NASDAQ OTC INDEX      TRANSPOR
KUTIL    NASDAQ OTC INDEX      UTILITIE
KNYSE    NEW YORK STK EXCHANGE COMPOSIT
KNYSF    NEW YORK STK EXCHANGE FINANCIA
KNYSI    NEW YORK STK EXCHANGE INDUSTRI
.
.
.
```

To retrieve and display the high, low, and closing index data using MAGIC, load the workspace 39 *MAGIC*:

```
DECIMALS 3
AUTOLABEL
AUTOTITLE
(⍳4) DAILY,DATED 21 1 80 TO 24 1 80   ⍝ set time frame
DISPLAY 'KAMEX' USOPTIONS 1 2 3       ⍝ retrieve and display high, low, closing
                                        index data
```

*KAMEX*

| | 21*JAN*80 | 22*JAN*80 | 23*JAN*80 | 24*JAN*80 |
|---|---|---|---|---|
| *HIGH* | 266.940 | 266.530 | 264.150 | 267.430 |
| *LOW* | 264.460 | 260.740 | 259.780 | 264.000 |
| *CLOSE* | 266.390 | 261.000 | 264.000 | 266.000 |

On-line documentation of the U.S. Options data base—including *MARKET INDICATOR* data— is available using 55 *RETRIEVE* by typing Δ*VIEWDOC* '*OPTIONUS*'.

Further inquiries are welcome, and should be addressed to your local Sharp representative or to George Bihus in Toronto.

## COURSE SCHEDULE

### INTRODUCTION TO APL

**Aberdeen**
(3 days)
March 5-7
May 21-23

**Atlanta**
(5 day)
March 10-14
June 16-20
September 15-29
December 8-12

**Coventry**
(3 day)
March 18-20
May 20-22
July 22-24

**Gloucester**
(3 day)
April 28-30
June 2-4

**London**
(3 days)
March 10-12
April 15-17
May 13-15
June 16-18

**Montreal**
(English)
March 11-13
May 13-15
(French)
April 15-17

**Ottawa**
(non-programmers)
April 1-3
June 3-5
(programmers)
March 4-6
May 6-8
July 8-10

**Rochester**
(6 day)
Feb.25-27/Mar.3-5
April 14-16/21-23
May 12-14/19-21
June 16-18/23-25

**Stockholm**
(Swedish)
March 3-5
March 17-19

**Toronto**
(3 day)
March 3-5
March 17-19
Mar.31-Apr.2
April 14-16
April 28-30
May 12-14
May 27-29
June 9-11
June 23-25

**Warrington**
(3 day)
April 9-11
June 25-27

**Washington D.C.**
(half-days)
March 3-6

**Zurich**
(German)
(APL Einfuhrung 1)
March 10-12
(Afbaukurs/Einf 2)
March 17-19

## INTERMEDIATE APL

**London**, March 25, April 24, May 28, June 10
**Montreal**, (English or French), March 25-27, May 27-29
**Ottawa**, May 20-21
**Rochester**, (4 days) March 20,21,27,28 and June 19,20,26,27
**Toronto**, March 10-12, April 21-23, June 2-4
**Zurich**, Feb. 25-27, March 17-19

## SEMINARS and SPECIAL COURSES

**Aberdeen**
Appreciation of APL, April 8, June 19

**Coventry** (half-day seminars)
Formatting with SHARP APL, July 12
Event Trapping, May 10
N- and B-tasks, June 14
Shared Variables, April 12
Using SHARP APL Files, March 8

**Edmonton**
Statistical Analysis in SHARP APL, March 6
MAGIC for Time Series Analysis, March 20,
Financial Analysis in SHARP APL, April 3

**Gloucester**
Appreciation of APL, March 3, May 12

**London**
Advanced APL, March 17, June 23
APL Review, March 24, April 23, May 27, June 9
Appreciation of APL, March 21, April 10,
May 8, June 13
Use a Terminal, June 5

**Montreal**, (English and French)
AIDS, March 4, May 20
Event Trapping, April 29
Planning System Design, May 6
Plotting, March 18
Saving Money with N- and B-tasks, April 22
Shared Variables, March 6, May 8
SNAP - A Network Analysis Package (halfday), April 8

**Ottawa**
Batch (Non-Terminal) Tasks, April 22
Box-Jenkins, June 19
Highspeed Printing, April 24, July 24
Input Validation (Idiot Proofing), April 23
MAGIC, March 18, June 17
Regression Analysis, March 20, July 22

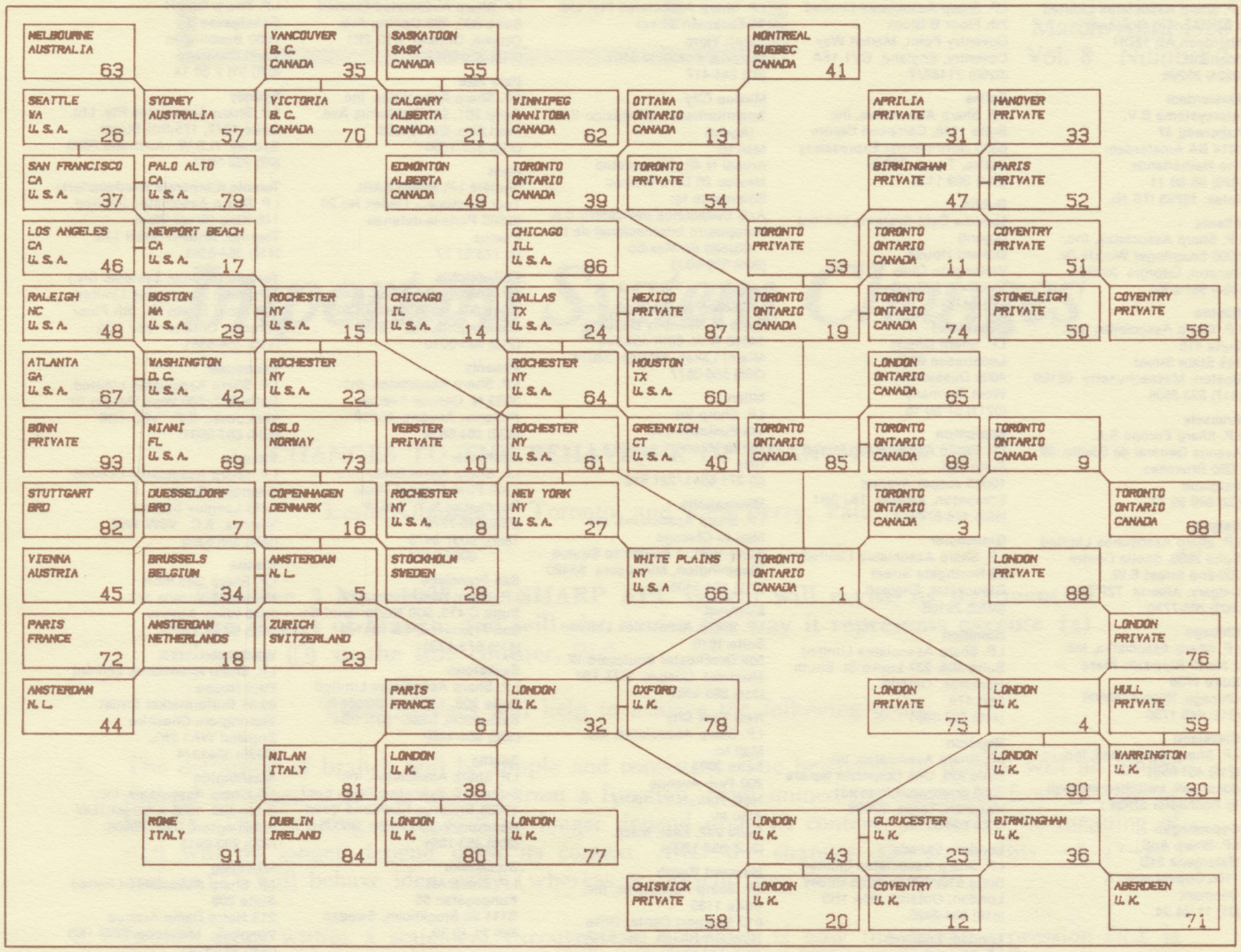
**Rochester**
Advanced Features Workshop, (4 half-days):
Package Variables, May 8
Batch APL, May 15
Event Trapping, May 22
Shared Variables, May 29
Advanced Methods Workshop (full days):
Systems Analysis and Design, April 3
Data Structures and File Design, April 10
Program Development, Debugging and Testing, April 17
Interactive Dialogues & Report Formatting, April 24
Efficient APL Programming, May 1

**Stockholm** (one-day seminars in Swedish)
Saving Money via N-tasks/B-tasks/HSPrints/Error Trapping,
Mar.24, Apr.21
File System & Shared Variables: Effective Design Techniques,
Mar.25, Apr.22
Better Programming through Use of Software in Libraries 1-1000,
Mar.26, Apr.23

**Toronto**
Advanced APL Coding Techniques for Actuarial Applications, May 1
Advanced APL Topics, April 8-10/14-17
(enroll for all or part of 7 sessions)
APL for Managers, Apr. 4, May 19, June 30
Batch Input Validation, April 10
Blessings of Booleans, April 15
Box-Jenkins, March 6, May 8
Data Representation, April 17
Data Structures/Shared Variables, April 8
Debugging Techniques, April 16
Event Trapping, (halfday), April 9
Forecasting Methods, March 7, May 9
MAGIC for Time Series Analysis, April 3, June 6
Regression Analysis, April 23, June 17
Report Formatting, April 25, June 20
Restartability & Secure Systems, April 14
Saving Money with N- and B-tasks, April 24, June 18
SUPERPLOT & MAGIC April 11, June 16
Text Editing, May 15-16
The Sharp System: A Practical Approach, March 13,
April 10, May 8, June 5

**Warrington**
Appreciation of APL, March 13, May 30

## Network Topology




## UPDATE

Label number: _____

☐ Please amend my mailing address as indicated.

☐ Add to your mailing list the following name(s).

☐ Send me a SHARP APL publications order form.

Name: _____

Co.: _____

Address: _____

Printed in Canada
February 1980

**I.P. Sharp Associates** **Head Office:** 145 King Street West, Toronto, Canada M5H 1J8 (416) 364-5361

## *International Branch Offices*

**Aberdeen**
I.P. Sharp Associates Limited
5 Bon Accord Crescent
Aberdeen AB 12DH
Scotland
(0224) 25298

**Amsterdam**
Intersystems B.V.
Kabelweg 47
1014 BA Amsterdam
The Netherlands
(020) 86 80 11
Telex: 18795 ITS NL

**Atlanta**
I.P. Sharp Associates, Inc.
5000 Snapfinger Woods Dr.
Decatur, Georgia 30035
(404) 987-2301

**Boston**
I.P. Sharp Associates, Inc.
Suite 415
148 State Street
Boston, Massachusetts 02109
(617) 523-2506

**Brussels**
I.P. Sharp Europe S.A.
Avenue General de Gaulle, 39
1050 Bruxelles
Belgique
(02) 649 99 77

**Calgary**
I.P. Sharp Associates Limited
Suite 2660, Scotia Centre
700-2nd Street S.W.
Calgary, Alberta T2P 2W2
(403) 265-7730

**Chicago**
I.P. Sharp Associates, Inc.
2 North Riverside Plaza
Suite 1736
Chicago, Illinois 60606
(312) 648-1730

**Cleveland**
I.P. Sharp Associates, Inc.
(216) 431-6861
(local call, switched through to Rochester office.)

**Copenhagen**
I.P. Sharp ApS
Østergade 24B
1100 Copenhagen K
Denmark
(01) 11 24 34

**Coventry**
I.P. Sharp Associates Limited
7th Floor B Block
Coventry Point, Market Way
Coventry, England CV1 1EA
(0203) 21486/7

**Dallas**
I.P. Sharp Associates, Inc.
Suite 1148, Campbell Centre
8350 Northcentral Expressway
Dallas, Texas 75206
(214) 369-1131

**Dublin**
Gamma Data Systems Limited
(Agent)
Dollard House
Wellington Quay
Dublin 2, Ireland
(01) 711877

**Düsseldorf**
I.P. Sharp GmbH
Leostrasse 62A
4000 Düsseldorf 11
West Germany
(0211) 57 50 16

**Edmonton**
I.P. Sharp Associates Limited
Suite 505
10065 Jasper Avenue
Edmonton, Alberta T5J 3B1
(403) 428-6744

**Gloucester**
I.P. Sharp Associates Limited
29 Northgate Street
Gloucester, England
(0452) 28106

**Hamilton**
I.P. Sharp Associates Limited
Suite 308, 237 Locke St. South
Hamilton, Ontario
L8P 4T4
(416) 527-3801

**Houston**
I.P. Sharp Associates, Inc.
Suite 925, One Corporate Square
2600 Southwest Freeway
Houston, Texas 77098
(713) 526-5275

**London, Canada**
I.P. Sharp Associates Limited
Suite 510, 220 Dundas Street
London, Ontario N6A 1H3
(519) 434-2426

**London, England**
I.P. Sharp Associates Limited
132 Buckingham Palace Road
London SW1W 9SA
England
(01) 730-0361

**Melbourne**
I.P. Sharp Associates Pty. Ltd.
36 Elizabeth Street
South Yarra
Victoria, Australia 3141
(03) 244-417

**Mexico City**
Teleinformatica de Mexico S.A.
(Agent)
Mail to:
Arenal N 40, Chimalistac
Mexico 20 D.F., Mexico
Shipments to:
A/C Despachos del Centro S.A.
Aeropuerto Internacional de la
Ciudad de Mexico
(905) 550-8033

**Miami**
I.P. Sharp Associates, Inc.
Suite D, Kennedy Building
14560 N.W. 60th Avenue
Miami Lakes, Florida 33014
(305) 556-0577

**Milan**
I.P. Sharp Srl
Via Eustacchi, II
20129 Milan
Italy
(2) 271 6541/221 612

**Minneapolis**
I.P. Sharp Associates, Inc.
Mail to Chicago
Suite 1371, 1 Appletree Square
Bloomington, Minnesota 55420
(612) 854-3405

**Montreal**
I.P. Sharp Associates Limited
Suite 1610
555 Dorchester Boulevard W.
Montreal, Quebec H2Z 1B1
(514) 866-4981

**New York City**
I.P. Sharp Associates, Inc.
Mail to:
Suite 2004
200 Park Avenue
New York, N.Y. 10017
Also at:
Suite 242, East. Mezz.
(212) 557-1200

**Newport Beach**
I.P. Sharp Associates, Inc.
Suite 1135
610 Newport Center Drive
Newport Beach, Ca. 92660
(714) 644-5112

**Oslo**
I.P. Sharp A/S
4. Etasje
Kronprinsesse Märthas Plass 1
Mail to:
Postboks 1470, Vika
Oslo 1, Norway
(02) 41 17 88

**Ottawa**
I.P. Sharp Associates Limited
Suite 600, 265 Carling Ave.
Ottawa, Ontario K1S 2E1
(613) 236-9942

**Palo Alto**
I.P. Sharp Associates, Inc.
Suite 201, 220 California Ave.
Palo Alto, Ca. 94306
(415) 327-1700

**Paris**
Société I.P. Sharp SARL
Tour Neptune - Cedex No.20
92086 Paris-la-defense
France
(1) 773 57 77

**Philadelphia**
I.P. Sharp Associates, Inc.
Suite 604, 437 Chestnut St.
Philadelphia, Pa. 19106
(215) 925-8010

**Phoenix**
I.P. Sharp Associates, Inc.
3033 N. Central Avenue
Phoenix, Arizona 85012
(602) 264-6819

**Rochester**
I.P. Sharp Associates, Inc.
1200 First Federal Plaza
Rochester, N.Y. 14614
(716) 646-7270
Telex: 0097 8473
0097 8474

**San Francisco**
I.P. Sharp Associates, Inc.
Suite C-415, 900 North Point St.
San Francisco, Ca. 94109
(415) 673-4930

**Saskatoon**
I.P. Sharp Associates Limited
Suite 208, 135 21st Street E.
Saskatoon, Sask. S7K 0B4
(306) 664-4480

**Seattle**
I.P. Sharp Associates, Inc.
Suite 217, Executive Plaza East
12835 Bellevue-Redmond Road
Bellevue, Washington 98005
(206) 453-1661

**Stockholm**
I.P. Sharp AB
Kungsgatan 65
S111 22 Stockholm, Sweden
(08) 21 10 19

**Stuttgart/Boeblingen**
I.P. Sharp GmbH
Schafgasse 3
7030 Boeblingen
West Germany
(070 31) 2 30 14

**Sydney**
I.P. Sharp Associates Pty. Ltd.
Suite 1342, 175 Pitt Street
Sydney, N.S.W., Australia 2000
(02) 232-5914

**Toronto (Corporate Headquarters)**
I.P. Sharp Associates Limited
145 King Street West
Toronto, Ontario M5H 1J8
(416) 364-5361

**Toronto (Special Systems Div.)**
I.P. Sharp Associates Limited
156 Front Street W., 5th Floor
Toronto, Ontario M5J 1G6
(416) 364-5361

**Vancouver**
I.P. Sharp Associates Limited
Suite 902, 700 West Pender St.
Vancouver, B.C. V6C 1G8
(604) 687-8991

**Victoria**
I.P. Sharp Associates Limited
Chancery Court
1218 Langley Street
Victoria, B.C. V8W 1W2
(604) 388-6365

**Vienna**
I.P. Sharp Ges.mbH
Rechte Wienzeile 5/3
1040 Wien, Austria
(222) 57 65 71

**Warrington**
I.P. Sharp Associates Limited
Paul House
89-91 Buttermarket Street
Warrington, Cheshire
England WA1 2NL
(0925) 50413/4

**Washington**
I.P. Sharp Associates, Inc.
Suite 400, 1835 K Street N.W.
Washington, D.C. 20006
(202) 293-2915

**Winnipeg**
I.P. Sharp Associates Limited
Suite 208
213 Notre Dame Avenue
Winnipeg, Manitoba R3B 1N3
(204) 947-1241

**Zurich**
I.P. Sharp A.G.
Badenerstrasse 141
8004 Zurich
Switzerland
(1) 241 52 42

## *SHARP APL Communications Network: Local Access Cities*

***APL OPERATOR VOICE (416) 363-2051 COMMUNICATIONS (416) 363-1832***

Local dial access is available in all locations listed above. The SHARP APL Communications Network also provides local dial access in:

• Ann Arbor • Birmingham • Buffalo • Dayton • Des Moines • Detroit • Ft. Lauderdale • Greene (NY) • Greenwich (Ct)
• Halifax • Hanover • Hartford • Hull • Kitchener • Laurel • Liverpool • Los Angeles • Lyndhurst • Manchester
• Oxford • Raleigh • Regina • Rome • Santa Ana • Syracuse • Towanda • Ukiah • White Plains (NY)

In the United States the SHARP APL Network is interconnected with the networks of TYMNET and TELENET to provide local dial access in more than 100 other cities.

The Newsletter is a regular publication of I.P. Sharp Associates. Contributions and comments are welcome and should be addressed to: Jeanne Gershater, I.P. Sharp Newsletter, 145 King Street West, Toronto, Canada M5H 1J8.

Jeanne Gershater, *Editor*

Ginger Kahn, *Assistant Editor*

# Technical Supplement–25

## BRANCHING IN SHARP APL, PART IV

Robert Metzger, IPSA Rochester

In the past three segments of this series, we have looked at four ways to avoid the use of the branch arrow:

1. Use boolean conditional statements
2. Use compression
3. Use the execute function
4. Use APL's built-in loops

Hopefully, you have gained a greater appreciation of the many ways you can solve a problem in APL. Perhaps, too, you have gained some new skills for developing your APL programs. In this segment, we, again, look at two more techniques for expressing program logic **without** using the branch arrow.

The fifth method for avoiding branching is to

### USE SUBFUNCTIONS

A subfunction is a function invoked by another function. Some people use the phrase, 'subroutine call' — meaning the same thing. In several other computer languages, subfunctions are invoked with instructions very similar to their equivalent of the branch instruction. This is to be expected. Both branching and invoking a subfunction cause a different set of instructions to be executed than those which immediately follow the original instruction. The difference is that the subfunction returns to the point of departure, whereas the branch does not.

How can you tell when a branch should really be a subfunction call? The most obvious symptom is if you are keeping track of where your program was before you do the branch. You would do this if you wanted to return to the place where the branch occurred. This is, of course, simulating a subfunction call using a branch statement and a variable. Why not simplify your program, and let APL keep track of where you were?

Another symptom of the need for a subfunction is when a number of different branch statements transfer control to the same function line. Often such branching patterns indicate that the segment of the function being branched to is an independent segment which could stand alone.

A third symptom of the need for a subfunction is when several non-trivial statements are repeated as a unit in the same function, or in several different functions. The advantage of isolating such segments is that you can debug them once, and rely on them thereafter. On the other hand, each repetition of the same set of statements must be tested independently.

Using subfunctions simplifies APL programming. Easier programming is one reason why we sometimes avoid using the branch arrow.

The sixth method for avoiding branching is to

USE RECURSION

A function is recursive if it invokes itself. The strategy is basically this: 'divide-and-conquer.' Recursive function logic often follows the pattern below:

Is the part of the problem at hand simple to solve?
If yes, then solve it.
If no, isolate a piece of the problem, and give that piece to the same program to attempt to solve it.

Obviously, it is possible to create an endless recursive sequence. If a recursive function is going to stop, it must have two properties:

1. At least one call in the cycle of subordinates must be conditional,

2. The exact values of input arguments may not repeat within recursive calls.

[Yourdon and Constantine, **Structured Design**, Prentice Hall, 1978, p. 289]

When should recursion be used in place of branching? The use of a loop, combined with a data structure called a stack, sometimes indicates a need for recursion. A stack is a data structure in which the last piece of information stored is the first to be retrieved (Last In, First Out). The State Indicator in the active workspace is a stack. The two functions listed below allow you to store and retrieve vectors (of equal length) in a global stack.

```
    ∇STORE VECTOR
[1]  ⍎(0=⎕NC 'STACK')/'STACK←(0,⍴VECTOR)⍴0'
[2]  STACK←VECTOR,[1]STACK
    ∇

    ∇VECTOR←FETCH
[1]  VECTOR←(1,¯1↑⍴STACK)↑STACK
[2]  STACK←1 0↓STACK
    ∇
```

If you need to manipulate a stack, and you are using a loop built with branches to do so, you might be better off using recursion. If you are writing statements like the functions above, you are probably using a stack.

You may want to consider recursion if you are working in an application area where recursive definitions are commonly useful. One such area is the definition of mathematical functions. Most people are familiar with the recursive definition of the factorial function.

```
    ∇Z←FAC X
[1]  →(X=0)/SIMPLE
[2]  Z←X×FAC X-1
[3]  →END
[4]  SIMPLE:Z←1
[5]  END:
    ∇
```

The factorial function is easily defined in this way. However, the factorial problem doesn't have one of the two properties that should be present, to really benefit from a recursive definition:

1. The problem is recursively defined — the subproblems are identical to the original problem;

2. The problem is not one dimensional (linear) — backups and restarts are necessary in solving the problem.

[Meyers, **Composite/Structured Design**, VanNostrand Reinhold, 1978, p. 63]

The second property is not present. The factorial problem is one dimensional — there is no need to go back to a previous value of the argument. The current value tells us all we need to know. Many more sophisticated mathematical functions, however, can benefit from a recursive definition.

A second area where recursion is often helpful is in symbolic manipulation. This includes natural language processing, formula manipulation, computer language processors, etc. An example is the function listed below. It will expand a ⎕*FMT* format string which includes nested repetition using parentheses. It produces an equivalent format which does not use nested repetition. This is useful if you need to do computations based upon a format string.

```
    ∇ FMT←EXPAND FMT;
MASK;NESTING;START;STOP;REPEAT;NUMBER;PHRASE;⎕IO
[1]   ⎕IO←1
[2]  ⍝ IS THERE WORK TO BE DONE ON THIS PIECE?
[3]  LOOP:→(~'('∊FMT)⍴END
[4]  ⍝ MASK OUT LITERAL TEXT BETWEEN '
[5]   MASK←FMT='''
[6]   MASK←MASK∨~≠\MASK
[7]  ⍝ DETERMINE THE LEVEL OF PARENTHESIS NESTING
[8]   NESTING←(MASK∧FMT=')')++\MASK× 1 ¯1 0['()'⍳FMT]
[9]  ⍝ IDENTIFY ONE PIECE TO WORK ON
[10]  START←FMT⍳'('
[11]  STOP←((FMT=')')∧(NESTING=NESTING[START]))⍳1
[12] ⍝ HOW MANY TIMES IS THIS PIECE TO BE REPEATED?
[13]  REPEAT←START-((⌽(START-1)↑FMT)∊'0123456789')⍳0
[14]  NUMBER←⍎REPEAT↓(START-1)↑FMT
[15] ⍝ SOLVE THIS PART OF THE PROBLEM WITH RECURSION
[16]  PHRASE←EXPAND START↓(STOP-1)↑FMT
[17] ⍝ REPEAT THIS PIECE AND PUT IT INTO THE SOLUTION
[18]  PHRASE←¯1↓(NUMBER×1+⍴PHRASE)⍴PHRASE,','
[19]  FMT←(REPEAT↑FMT),PHRASE,STOP↓FMT
[20]  →LOOP
[21] END:
    ∇
```

If this function was given the format

```
I2,X1,2(I5,3('|',F8.2),X1,4A1),F10.2'
```

it would produce the following equivalent format

```
I2,X1,I5,'|',F8.2,'|',F8.2,'|',F8.2,X1,4A1,I5,'|',F8.2,'|',
                                F8.2,'|',F8.2,X1,4A1,F10.2
```

Note that this problem does fit the two criteria mentioned above. In particular, it is necessary to backtrack and refer to previous parts of the problem.

A third area where recursion is often helpful is in manipulating special data structures, particularly trees and networks. Recursion is especially appropriate if the data is explicitly structured. Such data contains 'explicit structural information that relates, by reference, an element of data to other elements of data.'

[Yourdon and Constantine, p. 291]

An example of such data is a set of package variables in a file which describe a tree structure. Each package variable contains the names of the nodes which are subordinate to the node at hand (stored in a character matrix), and pointers to components which contain information about those subordinate nodes. Nodes which are end points (leaves) contain a negative pointer.

A recursive function which traverses the tree and prints the node names, is listed below. Successive levels are indented to indicate their node membership.

```
    ∇ NAME PRINTΔTREE COMPONENT;
PACKAGE;NAMES;POINTERS;CTR;LIMIT
[1]   INDENT←INDENT+5×0≠ρNAME
[2]  ⍝ IS THIS A TERMINAL NODE(LEAF)?
[3]   →(¯1=×COMPONENT)/LEAF
[4]  BRANCH:⎕←(INDENTρ' '),NAME
[5]   PACKAGE←⎕READ TREE,COMPONENT
[6]   'NAMES POINTERS' ⎕PDEF PACKAGE
[7]   CTR←0
[8]   LIMIT←ρPOINTERS
[9]  LOOP:CTR←CTR+1
[10]  →(LIMIT<CTR)/END
[11] ⍝ DESCEND THIS BRANCH
[12]  NAMES[CTR;] PRINTΔTREE POINTERS[CTR]
[13]  →LOOP
[14] END:→EXIT
[15] LEAF:⎕←(INDENTρ' '),NAME
[16] EXIT:INDENT←INDENT-5
    ∇
```

The function would be executed as follows:

```
'999 TREEFILE' ⎕STIE TREE←11
INDENT←0
'' PRINTΔTREE 1
```

A variety of problems can be expressed in terms of data structures like trees, which can be handled in a straightforward manner using recursion.

It is obvious from the above examples that recursion does not eliminate the need for branching. Recursion reduces the need for branching. It substitutes a more simple and elegant method which reduces the number of variables and logic statements required, by using function calls and by letting the APL processor manage the data stack.

In the next (and final) segment of this series, we will discuss how to use the branch arrow when it is the most appropriate means of expressing program logic.

All of the functions used as examples in this series can be found in workspace 999 *BRANCHING*, including alternative (non-recursive) definitions of *EXPAND* and *PRINTΔTREE*.

Your comments and questions regarding this series are welcomed and may be directed to the author at the Rochester office of I.P. Sharp Associates.

## *** CONTEST 9 RESULTS ***

Congratulations to all who participated in our frequency tabulation contest. The entrants and their solutions were:

| | |
|---|---|
| Andreas Werder, IPSA Zurich | *FTAB∆AWE*1/2 |
| Gary Beaver, IPSA Chicago | *FTAB∆BEAV*1/2 |
| Bruce Richardson, Toronto | *FTAB∆BRI*1 |
| David Alis | *FTAB∆DAL*1/2 |
| Dr. Joseph Specht, Heidelberg | *FTAB∆DJS*1/2/3 |
| Fabio Massimo, Milan | *FTAB∆FMAS*1 |
| Gene McDonnell, IPSA Palo Alto | *FTAB∆GMC*1 |
| Henri Brudzewsky, Denmark | *FTAB∆HENRI*1 |
| Roger Hui, Toronto | *FTAB∆HUI*1/2/3 |
| J. Henri Schueller, IPSA Calgary | *FTAB∆JHS*1/2 |
| Ken Pawulski, IPSA Toronto | *FTAB∆KTP*1 |
| Lloyd L. Ash, Trimac Consulting, Calgary | *FTAB∆LLA*1/2/3/4/5/6/7/8 |
| Mike Page, IPSA Toronto | *FTAB∆MPA*1/2 |
| Ray Chantler, Rowntree-Mackintosh Toronto | *FTAB∆RAYC*1/2/3 |
| Richard M. Heiberger, U. of Penn., Philadelphia | *FTAB∆RMH*1/2/3/4/5/6 |
| Wolfgang E. Fendt, Heidelberg | *FTAB∆WEF*0/1 |
| Bill Royds, IPSA Ottawa | *FTAB∆WGR*1/2/3/4/5/6 |
| Zeke Hoskin, IPSA Vancouver | *FTAB∆ZEKE*1/2/3/4/5/6 |

When completeness tests were run, we found that a few of the solutions submitted by Richard M. Heiberger and Bill Royds contained constructs still in the experimental stage on SHARP APL (i.e., general arrays). As a consequence, these solutions had to be withdrawn from the competition. The remaining entries were then tested with a 750-column *DATA* matrix and the *CONTROL* matrix from example 1 in the contest specifications. Solutions still functioning at this level were then subjected to timing tests. The execution times, expressed as multiples of the fastest entry, are tabulated as follows:

| Entry | Ranked Time | Entry | Ranked Time |
|---|---|---|---|
| *FTAB∆JHS*1 | 1.000 | *FTAB∆ZEKE*3 | 2.461 |
| *FTAB∆DJS*1 | 1.048 | *FTAB∆ZEKE*4 | 2.464 |
| *FTAB∆WEF*1 | 1.058 | *FTAB∆ZEKE*5 | 2.466 |
| *FTAB∆DAL*2 | 1.073 | *FTAB∆JHS*2 | 2.612 |
| *FTAB∆AWE*1 | 1.184 | *FTAB∆FMAS*1 | 2.811 |
| *FTAB∆MPA*1 | 1.645 | *FTAB∆MPA*2 | 2.895 |
| *FTAB∆HUI*3 | 1.811 | *FTAB∆AWE*2 | 3.009 |
| *FTAB∆HENRI*1 | 1.867 | *FTAB∆BRI*1 | 3.134 |
| *FTAB∆RMH*2 | 1.892 | *FTAB∆LLA*4 | 4.285 |
| *FTAB∆DJS*2 | 1.895 | *FTAB∆GMC*1 | 4.333 |
| *FTAB∆RMH*1 | 1.902 | *FTAB∆LLA*8 | 5.203 |
| *FTAB∆HUI*1 | 1.936 | *FTAB∆LLA*3 | 5.452 |
| *FTAB∆WEF*0 | 1.946 | *FTAB∆KTP*1 | 7.607 |
| *FTAB∆HUI*2 | 1.961 | *FTAB∆RAYC*3 | 24.756 |
| *FTAB∆DJS*3 | 2.015 | *FTAB∆RAYC*1 | 24.777 |
| *FTAB∆ZEKE*2 | 2.221 | *FTAB∆RAYC*2 | 24.815 |
| *FTAB∆RMH*3 | 2.262 | *FTAB∆LLA*6 | 29.191 |
| *FTAB∆ZEKE*6 | 2.444 | *FTAB∆LLA*5 | 30.367 |

The overall winner of contest 9 is *** **J. Henri Schueller** *** of our Calgary office. Congratulations! The non-IPSA winner is *** **Dr. Joseph Specht** *** of the Stiftung Rehabilitation Centre in Heidelberg, Germany. The winners will be contacted shortly regarding their prizes.

The winning entries may be viewed in workspace 999 *CONTEST*. Many thanks to all who participated.